EDA-Laboratorio 4

07/12/2014

Mikel Barcina
Jose Ángel Gumiel

Índice de contenido

1 Introducción

En este laboratorio tenemos volvemos a trabajar sobre la base de datos de actores y películas. Se nos pide que diseñemos unos algoritmos que sean capaces de obtener el grado medio de relaciones que une a dos actores cualesquiera, y obtener los actores que tengan mayor centralidad. El concepto de centralidad significa los actores que más aparecen al hallar el camino entre dos actores.

Como este es el último laboratorio, recopilaremos en este informe todo el código de la aplicación, así como toda la información relevante.

2 Diseño de las clases

Como hemos venido haciendo hasta ahora, todas las clases que empleamos tienen los atributos privados y son accesibles mediante getters y setters. Así mismo, también serán privados los métodos que sólo vayan a ser utilizados por una única clase.

Hemos usado toda la estructura del laboratorio 1 y 3, aunque hemos corregido algunos fallos que tuvimos. Desde un primer momento diseñamos el primer laboratorio con la estructura de Hash Map.

En el paquete tenemos 10 clases, además de otras 6 adicionales para pruebas.

Las enumeramos a continuación:

- Main: Esta es la clase principal. Tiene el lanzador de la aplicación. Esta clase contiene unos métodos que no deberían de estar ahí, si no que su lugar correcto sería ListaActoresPrincipal, sin embargo no lo hemos cambiado. Los métodos son los de "estanRelacionados" del laboratorio anterior, y hemos añadido los correspondientes a este laboratorio, que son "gradoRelaciones", "nodoCentral" y "hallarNodoCentral". Además de otros dos métodos que usamos como subprogramas para ayudarnos a reducir la complejidad del código y que sea más modular.
- Actor: La clase tiene nombre como identificador y también tiene el atributo miListaPeliculas. En los métodos tiene anadirPelicula, eliminarPelicula, imprimirInformacion y tieneElMismoNombre. Además de los getters y setters.

  En el tercer laboratorio hicimos cambios, añadimos un atributo de tipo ListaActores, a la que llamamos listaColegas. Esta lista contiene todos los actores con los que ha participado.

  Tenemos un método llamado añadirColegas, que dada una película recorre la lista de actores (reparto) y se añaden a la lista de colegas del actor.

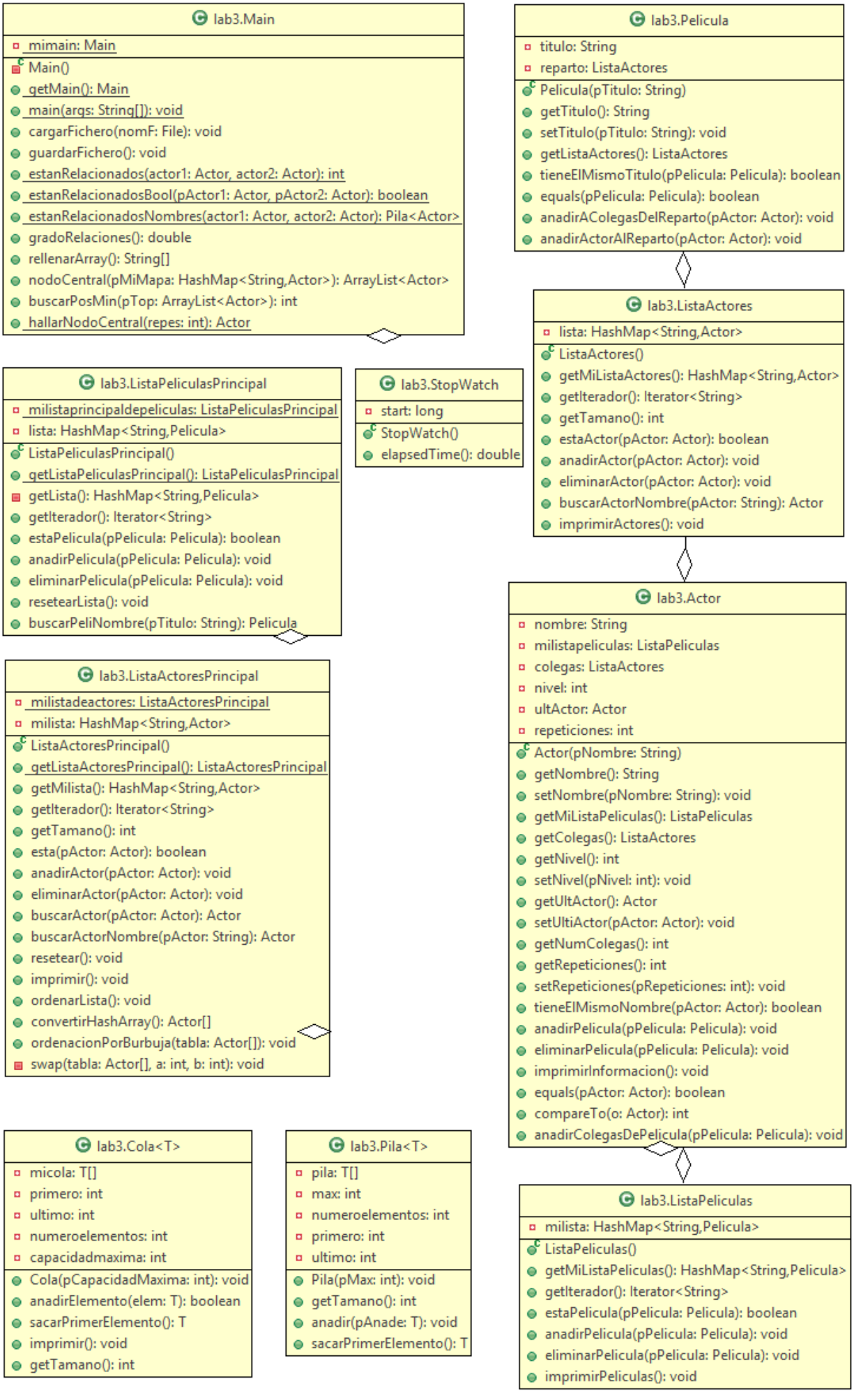
  También creamos el atributo nivel, con sus correspondientes getters y setters, que nos ayudaba a conocer la distancia a la hora de ejecutar el método estanRelacionados(). Somos conscientes de que es un atributo que nada tiene que ver con el actor, pero nos soluciona un problema de una forma bastante efectiva. Para este cuarto laboratorio hemos vuelto a repetir el mismo patrón para hallar el nodo central. Hemos añadido en esta clase un atributo privado de tipo entero llamado repeticiones. Cada vez que se traza un camino entre dos actores por el método "hallarNodoCentral", se leen los actores que aparecen en el camino y se les suma 1 en el atributo repeticiones, de forma que se sabe de forma fácil cuántas veces aparecen.
- Película: Aquí nos encontramos con una clase similar a la anterior. En un principio tenía el atributo título y el método tieneElMismoTítulo. En el laboratorio 3, tuvimos que añadir un atributo de tipo ListaActores, al que llamamos reparto. Esta lista contiene a todos los actores que han participado en esa película. A su vez, también tuvimos que crear un método llamado anadirActorAlReparto(), que añade un actor a la lista reparto.
- Cola: Esta clase implementa la estructura de tipo Cola. Estructura de datos que utilizamos para el método estanRelacionados, ya que tendremos una lista de elementos por examinar, en

la que tenemos que coger los elementos en orden. La estructura Cola es la que mejor se adapta, ya que tomamos el primer elemento y los nuevos los insertamos al final.

- Pila: Es una clase casi igual a la anterior, su diseño es muy parecido, sólo que la estructura de la Cola es de tipo FIFO y la de Pila es de tipo LIFO.
- ListaActores y ListaPelículas: Hemos usado los HashMaps. Llevan los métodos propios de listas, como el añadir, eliminar, esta e imprimir. La clase ListaActores es más completa, ya que incluye métodos para convertir el hashMap en Array, métodos para la ordenación (hemos usado el mergeSort) y buscarActor.
- ListaPeliculasPrincipal: Es una clase en la que se encuentra toda la colección de películas. Permite la búsqueda de películas.
- ListaActoresPrincipal: Es una clase en la que se encuentran todos los actores. Permite la búsqueda de actores, ya sea a la hora de mirar si un objeto de tipo Actor se encuentra en la lista o buscar un Actor por su nombre (String).
- StopWatch: Clase creada para analizar los tiempos de ejecución.

Además de estas clases tenemos también las otras que fueron creadas para hacer las pruebas y comprobaciones pertinentes. No pensamos que sea de interés detallarlas, y es por eso que nos limitaremos a poner el código de estas en su apartado correspondiente.

En la siguiente página se puede ver el diagrama de clases donde aparecen todas las clases mencionadas anteriormente y la totalidad de sus métodos y atributos, así como la relación existente entre ellas.


lab3.Main
mimain: Main
Main()
getMain(): Main
main(args: String[]): void
cargarFichero(nomF: File): void
guardarFichero(): void
estanRelacionados(actor1: Actor, actor2: Actor): int
estanRelacionadosBool(pActor1: Actor, pActor2: Actor): boolean
estanRelacionadosNombres(actor1: Actor, actor2: Actor): Pila<Actor>
gradoRelaciones(): double
rellenarArray(): String[]
nodoCentral(pMiMapa: HashMap<String,Actor>): ArrayList<Actor>
buscarPosMin(pTop: ArrayList<Actor>): int
hallarNodoCentral(repes: int): Actor
lab3.Pelicula
titulo: String
reparto: ListaActores
Pelicula(pTitulo: String)
getTitulo(): String
setTitulo(pTitulo: String): void
getListaActores(): ListaActores
tieneElMismoTitulo(pPelicula: Pelicula): boolean
equals(pPelicula: Pelicula): boolean
anadirAColegasDelReparto(pActor: Actor): void
anadirActorAlReparto(pActor: Actor): void
lab3.ListaActores
lista: HashMap<String,Actor>
ListaActores()
getMiListaActores(): HashMap<String,Actor>
getIterador(): Iterator<String>
getTamano(): int
estaActor(pActor: Actor): boolean
anadirActor(pActor: Actor): void
eliminarActor(pActor: Actor): void
buscarActorNombre(pActor: String): Actor
imprimirActores(): void
lab3.ListaPeliculasPrincipal
milistaprincipaldepeliculas: ListaPeliculasPrincipal
lista: HashMap<String,Pelicula>
ListaPeliculasPrincipal()
getListaPeliculasPrincipal(): ListaPeliculasPrincipal
getLista(): HashMap<String,Pelicula>
getIterador(): Iterator<String>
estaPelicula(pPelicula: Pelicula): boolean
anadirPelicula(pPelicula: Pelicula): void
eliminarPelicula(pPelicula: Pelicula): void
resetearLista(): void
buscarPeliNombre(pTitulo: String): Pelicula
lab3.StopWatch
start: long
StopWatch()
elapsedTime(): double
lab3.Actor
nombre: String
milistapeliculas: ListaPeliculas
colegas: ListaActores
nivel: int
ultActor: Actor
repeticiones: int
Actor(pNombre: String)
getNombre(): String
setNombre(pNombre: String): void
getMiListaPeliculas(): ListaPeliculas
getColegas(): ListaActores
getNivel(): int
setNivel(pNivel: int): void
getUltActor(): Actor
setUltiActor(pActor: Actor): void
getNumColegas(): int
getRepeticiones(): int
setRepeticiones(pRepeticiones: int): void
tieneElMismoNombre(pActor: Actor): boolean
anadirPelicula(pPelicula: Pelicula): void
eliminarPelicula(pPelicula: Pelicula): void
imprimirInformacion(): void
equals(pActor: Actor): boolean
compareTo(o: Actor): int
anadirColegasDePelicula(pPelicula: Pelicula): void
lab3.ListaActoresPrincipal
milistadeactores: ListaActoresPrincipal
milista: HashMap<String,Actor>
ListaActoresPrincipal()
getListaActoresPrincipal(): ListaActoresPrincipal
getMilista(): HashMap<String,Actor>
getIterador(): Iterator<String>
getTamano(): int
esta(pActor: Actor): boolean
anadirActor(pActor: Actor): void
eliminarActor(pActor: Actor): void
buscarActor(pActor: Actor): Actor
buscarActorNombre(pActor: String): Actor
resetear(): void
imprimir(): void
ordenarLista(): void
convertirHashArray(): Actor[]
ordenacionPorBurbuja(tabla: Actor[]): void
swap(tabla: Actor[], a: int, b: int): void
lab3.Cola<T>
micola: T[]
primero: int
ultimo: int
numeroelementos: int
capacidadmaxima: int
Cola(pCapacidadMaxima: int): void
anadirElemento(elem: T): boolean
sacarPrimerElemento(): T
imprimir(): void
getTamano(): int
lab3.Pila<T>
pila: T[]
max: int
numeroelementos: int
primero: int
ultimo: int
Pila(pMax: int): void
getTamano(): int
anadir(pAnade: T): void
sacarPrimerElemento(): T
lab3.ListaPeliculas
milista: HashMap<String,Pelicula>
ListaPeliculas()
getMiListaPeliculas(): HashMap<String,Pelicula>
getIterador(): Iterator<String>
estaPelicula(pPelicula: Pelicula): boolean
anadirPelicula(pPelicula: Pelicula): void
eliminarPelicula(pPelicula: Pelicula): void
imprimirPeliculas(): void

3 Descripción de las estructuras de datos principales

Para este laboratorio hemos usado dos estructuras de datos diferentes, Hash Map y Cola.

- HashMap: Es una estructura de datos claves (keys) con valores (values). La motivación para usar esta estructura es que ofrece un soporte muy eficiente para la búsqueda de datos. Se puede acceder a los elementos de la tabla con una clave generada. El funcionamiento del HashMap consiste en transformar la clave en un hash, un número que identifica la posición en la que se encuentra el valor deseado.

  Esta estructura gana utilidad cuando trabajamos con cantidades de información considerables, tal y como ocurre en nuestro caso.

- Cola: La particularidad de una estructura de datos de cola es el hecho de que sólo podemos acceder al primer y al último elemento de la estructura. Así mismo, los elementos sólo se pueden eliminar por el principio y sólo se pueden añadir por el final de la cola. Nos ha resultado especialmente útil, como hemos mencionado anteriormente, para tomar los actores por examinar.
- Pila: Estructura similar a la Cola, pero de tipo LIFO.
- Array: Una estructura de datos ya conocida. Es un vector que almacena objetos de un mismo tipo. El Array lo usamos para mostrar los actores con mayor centralidad. Usamos una estructura estática, es decir, el Array que usamos tiene un tamaño de 10 elementos.

4 Diseño e implementación de los métodos principales

En este apartado consideramos que los métodos principales son dos, el que mide la distancia y el que nos dice si están relacionados, es por ello que sólo nos centraremos en esos.

Clase Main:

- Método main(): Es el que ejecuta el programa principal. Consiste en un menú que nos permite escoger las opciones correspondientes. Damos 13 opciones:

```
1- Cargar fichero
2- Añadir un actor
3- Añadir una película
4- Ordenar la lista de actores
5- Guardar fichero
6- Imprimir colegas de un actor
7- Ver relación entre dos actores (distancia)
8- ¿Están relacionados los actores? (boolean)
9- Ver relación entre dos actores (distancia y nombres)
10- Hallar el grado de relaciones de los actores en la list
11- Mostrar Top 10 (Más nodos)\n"
12- top 10 (Nodo Central)
13- Salir
```

  Cada caso se trata de una manera independiente, llamando a los métodos y realizando las operaciones que se consideren oportunas.
  Sólo se podrá salir el programa cuando se seleccione la opción 13.

- Método cargarFichero(): Lanza una ventana que permite examinar la ubicación en la que tenemos guardado el archivo de texto. Una vez seleccionado, lo analiza y añade los actores y las películas a la lista.

- Método guardarFichero(): Una vez de que tenemos actores y películas en nuestro programa, esta opción nos permite exportarlo a un fichero de texto. La ruta que hemos definido es C:/actrices.txt. Este método puede dar problemas, ya que es posible que se necesiten permisos por parte del SO para escribir en esa ubicación.
- Método estanRelacionados(): Dados dos actores nos dice su relación entre ellos mediante un entero. Para este método utilizamos dos estructuras diferentes, una Cola para los elementos que aún están por examinar, y una lista de tipo Hash Map para los actores que ya están examinados.

  En un primer lugar seteamos los niveles a 1. A continuación añadimos el actor que queremos comparar a la cola porExaminar. Empieza el primer while, sólo se saldrá de él si el booleano enc es true o si la cola está vacía.

  Se compara si el primer elemento de la cola está en la lista de colegas del actor2. Si es así devuelve true y devolvemos el nivel. De lo contrario, cogemos los colegas del actor por examinar y los añadimos a la cola cambiando su nivel al nivel del actor anterior pero sumándole uno.

  Dado que si volvemos a ejecutar el programa de nuevo obtendríamos resultados incorrectos porque hemos hecho modificaciones en el atributo nivel de los actores, lo que hacemos es guardar la distancia en una variable para proceder a restaurar los actores a su estado original. En el caso de la cola cogemos el primer elemento, lo buscamos en la ListaActoresPrincipal y ponemos su nivel a cero. Con la lista de examinados hacemos lo mismo, pero como es un Hash Map tenemos que utilizar un iterador. Al final se quedan las listas vacías y los actores tal y como estaban.
- Método estanRelacionadosBool(): Como el primer método que diseñamos fue el que devolvía la distancia, este método que devuelve un valor booleano es muy sencillo. Ejecuta el método expuesto anteriormente, y si el valor es mayor que 0 devuelve True, afirmando que sí que existe una relación entre los dos actores.

  Añadir que en la especificación no decía qué es lo que tenía que devolver si comparamos la relación entre un actor y él mismo de nuevo. Lo incluimos en el caso general.
- Método estanRelacionadosNombres(): Este método era opcional en el laboratorio 3, en aquel entonces no lo implementamos, pero para este cuarto laboratorio sí que nos ha hecho falta, así que lo hemos tenido que añadir. Para mostrar los resultados en el orden correcto, hemos añadido en Actor una variable llamada ultimoActor, que nos muestra cual ha sido el anterior. Si llega al final, demostrando que están relacionados, vamos apilándolos uno detrás de otro. Con la estructura Pila conseguimos que esos actores se muestren en el orden correcto, desde el primero hasta el último.
- Método gradoRelaciones(): Determina la distancia media a la que se encuentra un actor de otro. Para ello se ejecuta el estanRelacionados() y se mide la distancia. Esto se repite las veces suficiente como para que podamos obtener un dato fiable. Consiste en sumar en una variable llamada “distancia” el resultado que obtenemos al ejecutar estanRelacionados() con dos actores aleatorios. Después dividimos la distancia entre el número de pruebas que hemos hecho (las veces que hemos llamado a ese subprograma). Obtenemos un número que es la media aritmética.
- Método nodoCentral(): Hemos pensado dos formas diferentes de hallar el nodo central, esta es una de ellas. Creemos que es muy probable que los actores que sean “nodos centrales”, es decir, los que más veces aparezcan al hallar la relación entre dos actores, van a ser los que más colegas tengan. Es por eso que para hallar los 10 actores con mayor centralidad hemos decidido buscar cuales son los actores con más colegas. Analizamos el Hash Map de la clase ListaActoresPrincipal y nos fijamos en los amigos que tiene cada actor. Los actores se guardan

en un Array de 10 posiciones, y siempre que se añade un actor nuevo, habrá que eliminar el Actor del Array que menos amigos tenga. Cuando finalice el programa tendremos una lista de los actores con más colegas, que podrá ser imprimida por pantalla.

- Método hallarNodoCentral: Esta es la otra alternativa, más fiable. Le decimos al usuario que dé el número de pruebas que se quieren hacer. Cuanto más grande sea el número, más preciso será el resultado, siempre que el número introducido no supere al número total de actores.

  En este caso, para hallar la centralidad llamamos al subprograma estanRelacionadosNombres(). Con la Pila que nos devuelve marcamos las apariciones de los actores, ya que escribimos en el atributo "repeticiones" del Actor en concreto. Una vez que se haya repetido el proceso las veces que el usuario haya marcado habrá que recorrer la lista de actores principal para mirar cuales han sido los actores que han aparecido más veces, y cuando se haya recorrido entera volver a restaurar el atributo "repeticiones" de cada Actor a 0.

Clase Actor:

- Método tieneElMismoNombre()

  Este método comprueba si dos actores tienen el mismo nombre o no y nos devuelve un boolean. En teoría no existen dos actores con el mismo nombre, luego, si encontramos que el resultado es true, será indicativo de que son la misma persona.

- Método anadirPelicula()

  Añade una película a "milistapeliculas", que es la lista de películas individual del actor.

- Método eliminarPelicula()

  Elimina un objeto de tipo Película de "milisapeliculas".

- Método imprimirInformacion()

  Muestra por pantalla el nombre del actor y su lista de películas.

Clase Pelicula:

- Método tieneElMismoTitulo()

  Es el equivalente al "tieneElMismoNombre()" de actor. Comprueba si dos películas tienen el mismo título. No obstante, las pruebas a realizar serán del mismo tipo.

  No hay ningún otro método destacable en esta clase.

Clase ListaActores:

Esta clase contiene métodos que no consideramos de interés para explicar. Son anadirActor(), eliminarActor() e imprimir(). Son casi idénticos a otros que hemos explicado anteriormente, y se van a repetir también en la clase ListaPeliculas.

- Método esta()

  Comprueba si un actor está en la Lista de Actores. Esto se hace a través de las claves del HashMap.

- Método buscarActor()

  Comprueba si el Actor está en la lista y de ser así devuelve un objeto de tipo Actor. Si no está devuelve null.

- Método resetear()

  Vacía la lista dejándola sin ningún objeto.

Clase ListaPeliculas:

Los métodos que aparecen en esta clase ya han sido también utilizados en ListaActores, por lo que no hay que resaltar ni explicar ningún método nuevo.

Clase ListaPeliculasPrincipal:

Ocurre lo mismo que con la clase anterior. No hay ningún método nuevo, sólo los habituales para el manejo de listas.

Clase StopWatch:

- Método elapsedTime()

  Muestra el tiempo que ha transcurrido desde el momento inicial hasta que se para el tiempo.

4.1 Alternativas examinadas

Ante la complejidad que presentaba hallar el nodo central, en un primer lugar planteamos un enfoque diferente. Pensamos que los actores con un mayor número de colegas iban a ser también los que más probabilidad tendrían de aparecer en un nodo central. Es por eso que decidimos de hacer el método basándonos en esa teoría.

Posteriormente se nos ocurrió una forma de hallar el nodo central basándonos en casos reales. Para eso pedimos que se nos de un número, que son las pruebas que se van a realizar para hallar el nodo central. Cuanto más alto sea el número, más ejecuciones se harán del método estanRelacionadosNombres() con actores aleatorios de la lista principal de actores. Apuntamos las veces que aparece cada actor, y posteriormente lo recorremos para poder decir qué 10 actores son los que más se repiten, es decir, qué 10 actores son los centrales.

En los últimos días, nos dimos cuenta de que deberíamos de haber creado una clase auxiliar que fuera una herencia de Actor para poner atributos no correspondientes al Actor, como son “repeticiones” o “nivel”. De este modo, los actores sólo tienen la información relevante a ellos.

5 Código

Hemos intentado poner el código de una forma que se adapte a la página, sin embargo, hay líneas de código que son excesivamente largas y nos es imposible adaptarlas mejor a este documento. Este problema resalta más en la clase Main.

5.1 Programa

A continuación mostramos la totalidad del código que hemos implementado, ordenado por clases:

5.1.1 Clase Main

```
package lab3;

import java.io.*;  //Importo el Paquete Entero
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class Main {
	//Atributos
	private static Main mimain = new Main();


	//Constructora
	private Main(){	}


	//Getters y Setters
	public static Main getMain(){
		return mimain;
	}


	//Otros Métodos

	public static void main(String[] args){
		boolean repetir;
		repetir = true;
		JOptionPane.showMessageDialog(null, "Bienvenido a la base virtual de cine");
		do{
			String entrada = JOptionPane.showInputDialog("¿Qué quieres hacer?\n"
					+ "1- Cargar fichero\n"
					+ "2- Añadir un actor\n"
					+ "3- Añadir una pelicula\n"
					+ "4- Ordenar la lista de actores\n"
					+ "5- Guardar fichero\n"
					+ "6- Imprimir colegas de un actor\n"
					+ "7- Ver relación entre dos actores (distancia)\n"
					+ "8- ¿Estan relacionados los actores? (boolean)\n"
					+ "9- Ver relación entre dos actores (distancia y nombres)\n"
					+ "10- Hallar el grado de relaciones de los actores en la lista\n"
					+ "11- Mostrar Top 10 (Más nodos)\n"
					+ "12- top 10 (nodo central)\n"
					+ "13- Salir");
			switch(entrada){
			case "1":
				//Opción 1:
				JFileChooser fc = new JFileChooser();
				fc.setCurrentDirectory(new File("."));
```

```
                                        fc.setDialogTitle("Elige un fichero");
                                        fc.setAcceptAllFileFilterUsed(false);
                                        if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) { //Comprueba que hemos seleccionado un
archivo
                                                StopWatch crono = new StopWatch();
                                                Main.getMain().cargarFichero(fc.getSelectedFile());
                                                System.out.println(crono.elapsedTime());
                                        } else {
                                                System.out.println("No seleccion ");
                                        }
                                        break;
                                case "2":
                                        Actor anadir = new Actor(JOptionPane.showInputDialog("Introduce el nombre del actor"));
                                        Pelicula anadepelicula;
                                        if(ListaActoresPrincipal.getListaActoresPrincipal().esta(anadir)){
                                                System.out.println("El actor ya se encuentra en la lista"); //ListaActores lo comprueba también
pero poniendo esto aqui te ahorras meterle peliculas para nada.
                                        }else{
                                                boolean quedanpeliculas = true;
                                                String respuesta;
                                                do{
                                                        respuesta = JOptionPane.showInputDialog("¿Quieres añadirle una película? (Si/No)");
                                                        if (respuesta.equalsIgnoreCase("si")){
                                                                anadepelicula = new Pelicula(JOptionPane.showInputDialog("Introduce el
titulo de la película"));
                                                                anadir.anadirPelicula(anadepelicula);
                                                        }else if (respuesta.equalsIgnoreCase("no")){
                                                                quedanpeliculas=false;
                                                        }else{
                                                                JOptionPane.showMessageDialog(null,   "La   respuesta   introducida   es
incorrecta. Por favor introduzca si o no");
                                                        }
                                                }while (quedanpeliculas);
                                                ListaActoresPrincipal.getListaActoresPrincipal().anadirActor(anadir);
                                        }
                                        break;
                                case "3":
                                        ListaPeliculasPrincipal.getListaPeliculasPrincipal().anadirPelicula(new
Pelicula(JOptionPane.showInputDialog("Introduce el titulo de la película")));
                                        break;
                                case "4":
                                        ListaActoresPrincipal.getListaActoresPrincipal().ordenarLista();
                                        break;
                                case "5":
                                        Main.getMain().guardarFichero();
                                        break;
                                case "6":
                                        String auuuux = JOptionPane.showInputDialog("Introduce el nombre del actor");
                                        Actor aaux = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(auuuux);
                                        if(aaux!=null){
                                                aaux.imprimirInformacion();
                                        }else{
                                                JOptionPane.showMessageDialog(null, "El actor introducido es incorrecto");
                                        }
                                        break;
                                case "7":
                                        String compara1 = JOptionPane.showInputDialog("Introduce el nombre del actor");
                                        String compara2 = JOptionPane.showInputDialog("Introduce el nombre del actor");
```

```
                                        Actor acompara1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(compara1);
                                        Actor acompara2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(compara2);
                                        int distancia = 0;
                                        if((acompara1!=null)&&(acompara2!=null)){
                                                distancia = estanRelacionados(acompara1,acompara2);

                                        }else{
                                                JOptionPane.showMessageDialog(null,"Uno de los actores introducidos no se encuentra en la
Lista de Actores");
                                        }
                                        System.out.println(distancia);
                                        break;
                                case "8":
                                        String compar1 = JOptionPane.showInputDialog("Introduce el nombre del actor");
                                        String compar2 = JOptionPane.showInputDialog("Introduce el nombre del actor");
                                        Actor acompar1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(compar1);
                                        Actor acompar2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(compar2);
                                        boolean relacionados = false;
                                        if((acompar1!=null)&&(acompar2!=null)){
                                                relacionados = estanRelacionadosBool(acompar1,acompar2);

                                        }else{
                                                JOptionPane.showMessageDialog(null,"Uno de los actores introducidos no se encuentra en la
Lista de Actores");
                                        }
                                        System.out.println(relacionados);
                                        break;

                                case "9":
                                        String comparas1 = JOptionPane.showInputDialog("Introduce el nombre del actor");
                                        String comparas2 = JOptionPane.showInputDialog("Introduce el nombre del actor");
                                        Actor acomparas1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(comparas1);
                                        Actor acomparas2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(comparas2);
                                        Pila<Actor> distancias=new Pila<Actor>(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
                                        if((acomparas1!=null)&&(acomparas2!=null)){
                                                distancias = estanRelacionadosNombres(acomparas1,acomparas2);
                                        }else{
                                                JOptionPane.showMessageDialog(null,"Uno de los actores introducidos no se encuentra en la
Lista de Actores");
                                        }
                                        try{
                                                while(distancias!=null){
                                                        System.out.println(distancias.sacarPrimerElemento().getNombre());
                                                }
                                        }catch(Exception e){}
                                        break;
                                case "10":
                                        System.out.println(Main.getMain().gradoRelaciones());
                                        break;
                                case "11":
        ArrayList<Actor>top=Main.getMain().nodoCentral(ListaActoresPrincipal.getListaActoresPrincipal().getMilista());
                                        break;

                                case"12":
                                        Actor one;
                                        String reps = JOptionPane.showInputDialog("Introduce cuantos casos analizar");
```

```
                                        int repes=Integer.parseInt(reps);
                                        if(repes<ListaActoresPrincipal.getListaActoresPrincipal().getTamano()){
                                                  one=hallarNodoCentral(repes);
                                                  System.out.println(one.getNombre());
                                        }
                                        break;
                              case "13":
                                                  repetir=false;
                                                  break;
                              default:
                                                  JOptionPane.showMessageDialog(null, "La opción introducida es incorrecta. Introduce un número del 1 al
5.");
                              }
                    }while(repetir==true);
          }



          public void cargarFichero(File nomF){
                    try{
                              FileReader fr = new FileReader(nomF);
                              @SuppressWarnings("resource")
                              BufferedReader br = new BufferedReader(fr);
                              String nombreactor;
                              String linea, tituloaux;
                              Actor ultimoactor = null;
                              Pelicula anadepelicula;
                              boolean todobien = true;
                              int ayuda = 0;
                              while ((linea=br.readLine())!=null) {
                                        ayuda=0;

                                        do{
                                                  try{
                                                            todobien=true;
                                                            if(linea.length()!=0){
                                                                      if(linea.substring(0,3).equals("\t\t\t")){
                                                                                if(ultimoactor!=null){
                                                                                          String[] sintabuladores = linea.split("\t");
                                                                                          String[] titulo = sintabuladores[3-ayuda].split("
*[(]+\\d+[)]");
                                                                                          tituloaux = titulo[0].replaceAll("[\"]","");


                                                                                          //Tenemos el título de la película
                                                                                          if(tituloaux.equals("")){
                                                                                                    todobien=false;
                                                                                                    ayuda++;
                                                                                          }else{
                                                                                                    anadepelicula                         =
ListaPeliculasPrincipal.getListaPeliculasPrincipal().buscarPeliNombre(tituloaux);

                                                                                                    if(anadepelicula==null){
                                                                                                              //No   está   en   la   Lista
Principal de Peliculas
                                                                                                              anadepelicula     =     new
Pelicula(tituloaux);
ListaPeliculasPrincipal.getListaPeliculasPrincipal().anadirPelicula(anadepelicula);
```

```
ultimoactor.anadirPelicula(anadepelicula);
                                                                                }else{
ultimoactor.anadirPelicula(anadepelicula);
                                                                                }
                                                                        }

                                                                }else{
                                                                        JOptionPane.showMessageDialog(null, "Ha ocurrido
un error");
                                                                }
                                                        }else{
                                                                String[] division = linea.split("\t");
                                                                nombreactor=division[0];

                                                                //Tenemos el nombre del actor
                                                                if(ayuda==0){
                                                                        ultimoactor                                     =
ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(nombreactor);
                                                                        if (ultimoactor==null){
                                                                                //No  está  en  la  Lista  Principal  de
Actores
                                                                                ultimoactor = new Actor(nombreactor);
ListaActoresPrincipal.getListaActoresPrincipal().anadirActor(ultimoactor);
                                                                        }
                                                                }

                                                                String[] titulo = division[2-ayuda].split(" *[(]+\\d+[)]");
                                                                tituloaux = titulo[0].replaceAll("\"","");

                                                               //Tenemos el titulo de la película
                                                                if(tituloaux.equals("\t")){
                                                                        todobien=false;
                                                                        ayuda++;
                                                                }else{
                                                                        anadepelicula                                   =
ListaPeliculasPrincipal.getListaPeliculasPrincipal().buscarPeliNombre(tituloaux);
                                                                        if(anadepelicula==null){
                                                                                anadepelicula          =          new
Pelicula(tituloaux);
ListaPeliculasPrincipal.getListaPeliculasPrincipal().anadirPelicula(anadepelicula);
ultimoactor.anadirPelicula(anadepelicula);
                                                                        }else{
ultimoactor.anadirPelicula(anadepelicula);
                                                                        }
                                                                }
                                                        }
                                                }
                                        }catch(ArrayIndexOutOfBoundsException ae){
                                                //A  veces  en  vez  de  salir  actriz\t\tpelicula  solo  hay  un  \t.  Lo
solucionamos aqui:
                                                todobien=false;
```

```
                                                                ayuda++;
   ListaActoresPrincipal.getListaActoresPrincipal().eliminarActor(ultimoactor); //Para evitar duplicados
                                                }
                                        }while(!todobien);
                                }
            }catch(IOException e) {
                        e.printStackTrace();
            }
   }


   public void guardarFichero(){
            FileWriter fichero = null;
PrintWriter pw = null;
try
{
    fichero = new FileWriter("C:\\Documents and Settings\\euitibi\\Escritorio\\actrices.txt");
    pw = new PrintWriter(fichero);
    String auxAct, auxPel;

    Iterator<String> itrAct = ListaActoresPrincipal.getListaActoresPrincipal().getIterador();
    Iterator<String> itrPel;

    while(itrAct.hasNext()){
            auxAct = itrAct.next();
            pw.println(auxAct);
            itrPel = ListaActoresPrincipal.getListaActoresPrincipal().getMilista().get(auxAct).getMiListaPeliculas().getIterador();
            while(itrPel.hasNext()){
                        auxPel=itrPel.next();
                        pw.println("--> "+auxPel);
            }
    }

} catch (Exception e) {
    e.printStackTrace();
}

finally {
   try {
            if (null != fichero)
               fichero.close();
   } catch (Exception e2) {
      e2.printStackTrace();
   }
}
   }


   public static int estanRelacionados(Actor actor1, Actor actor2){
            boolean enc = false;
            Pila<String> porExaminar = new Pila<String>(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
            ListaActores examinados = new ListaActores();
            porExaminar.anadir(actor1.getNombre());
            Actor actoraexaminar = new Actor("Actor por Defecto");
            Iterator<String> iteradordecolegas;
```

```
		String saux;
		Actor aaux;
		int nivel = 1;
		actor1.setNivel(nivel);//Porque si lo encuentra de primeras devuelve 0.

		while((enc==false)&&(porExaminar.getTamano()>0)){
			saux = porExaminar.sacarPrimerElemento();
			actoraexaminar = ListaActoresPrincipal.getListaActoresPrincipal().getMilista().get(saux);
			if(actoraexaminar.getColegas().estaActor(actor2)){
				enc=true;
			}else{
				iteradordecolegas = actoraexaminar.getColegas().getIterador();
				nivel=actoraexaminar.getNivel()+1;
				while(iteradordecolegas.hasNext()){
					saux = iteradordecolegas.next();
					aaux = actoraexaminar.getColegas().getMiListaActores().get(saux);
					if((aaux!=null)&&(!examinados.estaActor(aaux))){
						aaux.setNivel(nivel);
						porExaminar.anadir(aaux.getNombre());
						examinados.anadirActor(aaux);
					}
				}
			}
		}

		//-----------------------------------------------------------------------------------------

		int resultado = 0;
		if(enc){
			resultado=actoraexaminar.getNivel();
		}

		while(porExaminar.getTamano()!=0){
			saux = porExaminar.sacarPrimerElemento();
			Actor a = ListaActoresPrincipal.getListaActoresPrincipal().getMilista().get(saux);
			ListaActoresPrincipal.getListaActoresPrincipal().buscarActor(a).setNivel(0);
			//Esto borra la lista de actores porExaminar y ademas restaura su valor a 0.
		}
		while(examinados.getIterador().hasNext()){
			String a1=examinados.getIterador().next();
			ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(a1).setNivel(0);
			examinados.eliminarActor(examinados.buscarActorNombre(a1));
			/*Hace lo mismo con los examinados. Esto es porque si lo dejamos igual y
			volvemos a hacer otra búsqueda, el resultado no sería correcto, ya que
			tendrían niveles asignados.*/
		}

		return resultado;
	}

	public static boolean estanRelacionadosBool(Actor pActor1, Actor pActor2){
		boolean enc=false;
		if(estanRelacionados(pActor1, pActor2)>0){
			enc=true;
		}
```

```
		return enc;
}


public static Pila<Actor> estanRelacionadosNombres(Actor actor1, Actor actor2){
		boolean enc = false;
		Pila <Actor> camino=new Pila<Actor>(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
		Pila<Actor> porExaminar = new Pila<Actor>(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
		ListaActores examinados = new ListaActores();
		porExaminar.anadir(actor1);
		Actor actoraexaminar = new Actor("Actor por Defecto");
		Iterator<String> iteradordecolegas;
		String saux;
		Actor aaux;
		int nivel = 1;
		actor1.setNivel(nivel);//Porque si lo encuentra de primeras devuelve 0

		while((enc==false)&&(porExaminar.getTamano()>0)){
			actoraexaminar = porExaminar.sacarPrimerElemento();
			if(actoraexaminar.getColegas().estaActor(actor2)){
				enc=true;
			}else{
				iteradordecolegas = actoraexaminar.getColegas().getIterador();
				nivel=actoraexaminar.getNivel()+1;
				String ultiActorNom=actoraexaminar.getNombre();
				Actor ultiActor=ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(ultiActorNom);
				while(iteradordecolegas.hasNext()){
					saux = iteradordecolegas.next();
					aaux = actoraexaminar.getColegas().getMiListaActores().get(saux);
					if((aaux!=null)&&(!examinados.estaActor(aaux))){
						aaux.setNivel(nivel);
						aaux.setUltiActor(ultiActor);
						porExaminar.anadir(aaux);
						examinados.anadirActor(aaux);
					}
				}
			}
		}
		int resultado=actoraexaminar.getNivel();
		if (resultado!=0){
			Actor ulti=actoraexaminar;
			while (ulti!=actor1){
				camino.anadir(ulti);
				ulti=ulti.getUltActor();
			}
		}

		while(porExaminar.getTamano()!=0){
			Actor a=porExaminar.sacarPrimerElemento();
			ListaActoresPrincipal.getListaActoresPrincipal().buscarActor(a).setNivel(0);
			ListaActoresPrincipal.getListaActoresPrincipal().buscarActor(a).setUltiActor(null);
			//Esto borra la lista de actores porExaminar, restaura su valor a 0 y restaura ultiActor.
		}
		while(examinados.getIterador().hasNext()){
			String a1=examinados.getIterador().next();
```

```
                    ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(a1).setNivel(0);
                    ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(a1).setUltiActor(null);
                    examinados.eliminarActor(examinados.buscarActorNombre(a1));
                    /*Hace lo mismo con los examinados. Esto es porque si lo dejamos igual y
                    volvemos a hacer otra búsqueda, el resultado no sería correcto, ya que
                    tendrían niveles asignados.*/
          }

          return camino;
}

public double gradoRelaciones(){
          double gr = 0;
          int numeropruebas = ListaActoresPrincipal.getListaActoresPrincipal().getTamano();

          if(numeropruebas>500){
                    numeropruebas=500;
          }

          boolean fin = false;
          String[] arraynombres = new String[ListaActoresPrincipal.getListaActoresPrincipal().getTamano()];
          arraynombres = rellenarArray();
          int r1;
          int r2;
          double distancia = 0;
          double distanciaantigua = 0;
          int cont = 0; //Borrar tras las pruebas
          Random rg = new Random();
          Actor a1;
          Actor a2;

          while(!fin){

                    for (int i=0; i<numeropruebas; i++){
                              r1 = rg.nextInt(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
                              r2 = rg.nextInt(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
                              a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(arraynombres[r1]);
                              a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(arraynombres[r2]);
                              distancia += estanRelacionados(a1, a2);
                    }

                    distancia /= numeropruebas;

                    //Borrar tras las pruebas (o no)
                    cont++;
                    System.out.println("Vuelta número: "+cont);
                    System.out.println(distancia);
                    System.out.println(distanciaantigua);
                    System.out.println(fin);


                    if((distancia-0.05<distanciaantigua)&&(distanciaantigua<distancia+0.05)){
                              fin = true;
                              gr = distancia;
                    }else{
```

```
                        numeropruebas*=2;
                        distanciaantigua=distancia;
                        distancia = 0;
                }
        }

        return gr;
}

public String[] rellenarArray(){
        String[] miarray = new String[ListaActoresPrincipal.getListaActoresPrincipal().getTamano()];
        int cont = 0;

        Object[] arraydellaves = ListaActoresPrincipal.getListaActoresPrincipal().getMilista().keySet().toArray();
        for(Object a :arraydellaves){
                miarray[cont] = a.toString();
                cont++;
        }


        return miarray;
}

/*Este metodo define la centralidad como los actores que mas colegas tienen.
Es decir, devuelve una lista con los 10 actores que mas colegas tienen.*/
public ArrayList<Actor> nodoCentral(HashMap<String,Actor> pMiMapa) {
        ArrayList<Actor> topActores = new ArrayList<Actor>();
        Iterator<String> it = ListaActoresPrincipal.getListaActoresPrincipal().getIterador();
        for(int i=0; i<10;i++){
                //Suponemos que trabajamos con una lista de actores grande, asi que siempre tendrá más de 10 elementos.
                ListaActoresPrincipal.getListaActoresPrincipal().getIterador();
                String pActorNom=it.next();
                Actor pActor=ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(pActorNom);
                topActores.add(pActor);
                //Hasta aqui llenamos una lista con los 10 primeros actores.
        }

        Iterator<String> itr = ListaActoresPrincipal.getListaActoresPrincipal().getIterador();

        while(itr.hasNext()){
                String pActorNom=itr.next();
                Actor pActor=ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(pActorNom);
                int min=this.buscarPosMin(topActores);
                if(pActor.getNumColegas()>topActores.get(min).getNumColegas()){
                        topActores.add(min, pActor);
                }
        }
        //Metodo para imprimir el top 10 de actores
        for (int j=0; j < 10; j++) {
                Actor aux = topActores.get(j);
                System.out.println(aux.getNombre());
        }
        //Orden constante, 10 iteraciones, la longitud del array;
        return topActores;
}
```

```
public int buscarPosMin(ArrayList<Actor>pTop){
		int pos=0;
		int i=0;
		while(i<10) {
		    if (pTop.get(i).getNumColegas()<=pTop.get(pos).getNumColegas()) {
		        pos=i;
		    }
		    i++;
		}
		return pos;
}

/*Este otro metodo halla la centralidad basandose en las veces que un actor aparece al hallar una relacion.
 * Se pide al usuario que de un numero menor o igual que el numero de actores en lista, cuanto más grande, más fiable será.
 * Se cogen actores aleatorios y se hacen caminos. Por cada aparición de un actor se le anota.
 * Finalmente se mira qué actores han aparecido más veces.
 */
public static Actor hallarNodoCentral(int repes){
		int i=0;
		Pila<Actor> miPila=new Pila<Actor>(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
		while(i<repes){
			Random gen1 = new Random();
			Random gen2 = new Random();
			Object[] values = ListaActoresPrincipal.getListaActoresPrincipal().getMilista().values().toArray();
			Actor randomActor1 = (Actor)values[gen1.nextInt(values.length)];
			Actor randomActor2 = (Actor)values[gen2.nextInt(values.length)];
			//Esto coge un actor aleatorio de la lista de actores principal.
			Pila<Actor> miPilaAux=new Pila<Actor>(ListaActoresPrincipal.getListaActoresPrincipal().getTamano());
			miPilaAux=estanRelacionadosNombres(randomActor1, randomActor2);
			//Ahora apilamos los actores resultantes en la pila principal.
			while(miPilaAux.getTamano()!=0){
				miPila.anadir(miPilaAux.sacarPrimerElemento());
			}
			i++;
		}
		//Ahora tenemos una pila con todos los actores, hay que apuntar sus repeticiones.
		while(miPila.getTamano()!=0){
			Actor aux=miPila.sacarPrimerElemento();
			aux.setRepeticiones(aux.getRepeticiones()+1);
		}
		//Ahora hemos seteado las veces que un actor aparece repetido cuando hayamos el camino entre dos actores diferentes.
		//Es hora de recorrer el HashMap con el iterador y mirar quien es el actor que en más caminos aparece.
		String topNom=ListaActoresPrincipal.getListaActoresPrincipal().getIterador().next();
		Actor top=ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(topNom);
		//inicializamos top
		while(ListaActoresPrincipal.getListaActoresPrincipal().getIterador().hasNext()){
			String a1=ListaActoresPrincipal.getListaActoresPrincipal().getIterador().next();
			Actor ac1=ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(a1);
			if(ac1.getRepeticiones()>top.getRepeticiones()){
				top=ac1;
			}
		}
		//Ahora hay que restaurar todos los valores de repeticiones a 0
		while(ListaActoresPrincipal.getListaActoresPrincipal().getIterador().hasNext()){
```

```
                        String aux=ListaActoresPrincipal.getListaActoresPrincipal().getIterador().next();
                        ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre(aux).setRepeticiones(0);
                }
                return top;
        }
}
```

5.1.2 Clase Actor

```
package lab3;

import java.util.Iterator;

public class Actor{
	//Atributos
	private String nombre;
	private ListaPeliculas milistapeliculas;
	private ListaActores colegas;
	private int nivel; //Mide distancia en el estan relacionados
	private Actor ultActor;
	private int repeticiones; //Cada vez que el actor aparece en un camino, se suma
1.

	//Constructora
	public Actor(String pNombre){
		this.nombre=pNombre;
		this.milistapeliculas = new ListaPeliculas();
		this.colegas=new ListaActores();
		this.nivel = 0;
		this.ultActor=null;
	}


	//Getters y Setters
	public String getNombre() {
		return nombre;
	}

	public void setNombre(String pNombre) {
		this.nombre = pNombre;
	}

	public ListaPeliculas getMiListaPeliculas(){
		return this.milistapeliculas;
	}

	public ListaActores getColegas(){
		return this.colegas;
	}

	public int getNivel(){
		return this.nivel;
	}

	public void setNivel(int pNivel){
		this.nivel=pNivel;
	}

	public Actor getUltActor(){
		return this.ultActor;
	}
```

```
public void setUltiActor(Actor pActor){
	this.ultActor=pActor;
}

public int getNumColegas(){
	int num=0;
	Iterator<String> it = this.getColegas().getIterador();
	while(it.hasNext()){
		it.next();
		num++;
	}
	return num;
}

public int getRepeticiones(){
	return this.repeticiones;
}

public void setRepeticiones(int pRepeticiones){
	this.repeticiones=pRepeticiones;
}

//Otros Metodos
public boolean tieneElMismoNombre(Actor pActor){
	try{
		if(pActor.getNombre().equals(this.getNombre())){
			return true;
		}else
			return false;
	}
	catch (Exception e){
		return false;
	}
}

public void anadirPelicula(Pelicula pPelicula){
	try{
		 if(pPelicula.getListaActores().getTamano()==0){
			this.getMiListaPeliculas().anadirPelicula(pPelicula);
			pPelicula.getListaActores().anadirActor(this);
		 }else{
			pPelicula.anadirAColegasDelReparto(this);
			this.anadirColegasDePelicula(pPelicula);
			pPelicula.anadirActorAlReparto(this);
			this.getMiListaPeliculas().anadirPelicula(pPelicula);
		 }
	}catch(Exception e){
		System.out.println("La pelicula introducida no es válida");
	}
}

public void eliminarPelicula(Pelicula pPelicula){
	this.getMiListaPeliculas().eliminarPelicula(pPelicula);
}

public void imprimirInformacion(){
	System.out.println("Nombre: "+this.getNombre());
	System.out.println("Peliculas:");
	this.getMiListaPeliculas().imprimirPeliculas();
	System.out.println("Colegas:");
```

```
			this.colegas.imprimirActores();
		}

		public boolean equals(Actor pActor){
			return this.tieneElMismoNombre(pActor);
		}


		public int compareTo(Actor o) {
			return this.getNombre().compareTo(o.getNombre());
		}

		public void anadirColegasDePelicula(Pelicula pPelicula){
			Iterator <String> it=pPelicula.getListaActores().getIterador();
			while(it.hasNext()){
				String aux;
				Actor aaux;
				aux=it.next();
				if (aux!=null){
					aaux=pPelicula.getListaActores().buscarActorNombre(aux);
					if ((aaux!=null)&&(!aux.equals(this.getNombre()))){
						this.getColegas().anadirActor(aaux);
					}
				}
			}
		}
}
```

5.1.3 Clase Película

```
package lab3;

import java.util.Iterator;

public class Pelicula {
	//Atributos
	private String titulo;
	private ListaActores reparto;

	//Constructora
	public Pelicula(String pTitulo){
		this.titulo=pTitulo;
		this.reparto=new ListaActores();
	}


	//Getters y Setters
	public String getTitulo(){
		return this.titulo;
	}

	public void setTitulo(String pTitulo){
		this.titulo=pTitulo;
	}

	public ListaActores getListaActores(){
		return this.reparto;
	}


	//Otros Metodos
	public boolean tieneElMismoTitulo(Pelicula pPelicula){
		try{
			if(pPelicula.getTitulo().equals(this.getTitulo())){
				return true;
```

```
                }else
                        return false;
        }
        catch (Exception e){
                return false;
        }
    }

    public boolean equals(Pelicula pPelicula){
        return this.tieneElMismoTitulo(pPelicula);
    }

    public void anadirAColegasDelReparto(Actor pActor){
        try{
                Iterator<String> it = reparto.getIterador();
                while(it.hasNext()){
                        String saux = it.next();
                        if(!saux.equals(pActor.getNombre())){
                                Actor aaux = reparto.getMiListaActores().get(saux);
                                aaux.getColegas().anadirActor(pActor);
                        }
                }
        }catch(Exception e){
                System.out.println("El actor introducido no es válido");
        }
    }

    public void anadirActorAlReparto(Actor pActor){
        if(!this.getListaActores().getMiListaActores().containsKey(pActor.getNombre())){
                this.getListaActores().anadirActor(pActor);
        }
    }
}
```

5.1.4 Clase ListaActores

```
package lab3;

import java.util.HashMap;
import java.util.Iterator;

public class ListaActores {

    private HashMap<String, Actor> lista;

    public ListaActores(){
        this.lista = new HashMap<String, Actor>();
    }

    //Getters y Setters
    public HashMap<String,Actor> getMiListaActores(){
        return lista;
    }

    public Iterator<String> getIterador(){
        return this.getMiListaActores().keySet().iterator();
    }

    public int getTamano(){
        return this.lista.size();
    }


    //Otros Métodos
    public boolean estaActor(Actor pActor){
        try{
                return this.getMiListaActores().containsKey(pActor.getNombre());
        }catch(NullPointerException e){
                System.out.println("El Actor que estás buscando no se encuntra en la lista");
```

```
                        return false;
                }
        }

        public void anadirActor(Actor pActor){
                 if(pActor!=null){
                        if(!this.estaActor(pActor)){
                                 if(!ListaActoresPrincipal.getListaActoresPrincipal().esta(pActor)){
                                         ListaActoresPrincipal.getListaActoresPrincipal().anadirActor(pActor);
                                         this.getMiListaActores().put(pActor.getNombre(),pActor);
                                 }else{
                                         this.getMiListaActores().put(pActor.getNombre(),pActor);
                                 }
                        }
                }
        }

        public void eliminarActor(Actor pActor){
                if(pActor!=null){
                        if(estaActor(pActor)){
                                this.getMiListaActores().remove(pActor.getNombre());
                        }else{
                                System.out.println("La película no se encuentra en la lista");
                        }
                }
        }

        public Actor buscarActorNombre(String pActor){
                //Devuelve el Actor si está y sino devuelve null
                return this.getMiListaActores().get(pActor);
        }

        public void imprimirActores(){
                Iterator<String> it = this.getIterador();
                while(it.hasNext()){
                        System.out.println("->"+it.next());
                }
        }
}
```

5.1.5 Clase ListaPeliculas

```
package lab3;

import java.util.HashMap;
import java.util.Iterator;

public class ListaPeliculas {
        //Atributos
        private HashMap<String, Pelicula> milista;

        //Constructora
        public ListaPeliculas(){
                this.milista = new HashMap<String, Pelicula>();
        }


        //Getter y Setters
        public HashMap<String, Pelicula> getMiListaPeliculas(){
                return this.milista;
        }

        public Iterator<String> getIterador(){
                return this.getMiListaPeliculas().keySet().iterator();  //Esto crea un iterador de keys o llaves

//en nuestro caso titulos de peliculas.
        }


        //Otros Metodos
        public boolean estaPelicula(Pelicula pPelicula){
                try{
                        String pTitulo = pPelicula.getTitulo();
                        return this.getMiListaPeliculas().containsKey(pTitulo);
                }catch(NullPointerException e){
                        return false;
```

```
		}
	}

	public void anadirPelicula(Pelicula pPelicula){
		 if(pPelicula!=null){
			if(!this.estaPelicula(pPelicula)){
				 if(!ListaPeliculasPrincipal.getListaPeliculasPrincipal().estaPelicula(pPelicula)){
					ListaPeliculasPrincipal.getListaPeliculasPrincipal().anadirPelicula(pPelicula);
					this.getMiListaPeliculas().put(pPelicula.getTitulo(),pPelicula);
				 }else{
					this.getMiListaPeliculas().put(pPelicula.getTitulo(),pPelicula);
				 }
			}
		}
	}

	public void eliminarPelicula(Pelicula pPelicula){
		if(pPelicula!=null){
			if(estaPelicula(pPelicula)){
				this.getMiListaPeliculas().remove(pPelicula.getTitulo());
			}else{
				System.out.println("La película no se encuentra en la lista");
			}
		}
	}

	public void imprimirPeliculas(){
		Iterator<String> it = this.getIterador();
		while(it.hasNext()){
			System.out.println("->"+it.next());
		}
	}

}
```

5.1.6 Clase ListaActoresPrincipal

```
package lab3;

import java.util.HashMap;
import java.util.Iterator;

//import javax.swing.JOptionPane;


public class ListaActoresPrincipal {
	//Atributos
	private static ListaActoresPrincipal milistadeactores = new ListaActoresPrincipal();
	private HashMap<String, Actor> milista;


	//Constructora
	public ListaActoresPrincipal(){
		milista = new HashMap<String, Actor>();
	}


	//Getters y Setters
	public static ListaActoresPrincipal getListaActoresPrincipal(){
		return milistadeactores;
	}

	public HashMap<String,Actor> getMilista(){
		return this.milista;
	}

	public Iterator<String> getIterador(){
		return this.getMilista().keySet().iterator();
	}

	public int getTamano(){
		return this.getMilista().size();
	}
```

```
//Otros Metodos
	public boolean esta(Actor pActor){
		return this.getMilista().containsKey(pActor.getNombre());
	}

	public void anadirActor(Actor pActor){
		if(esta(pActor)){
			System.out.println("El actor ya se encuentra en la lista");
		}else{
			getMilista().put(pActor.getNombre(), pActor);
		}
	}

public void eliminarActor(Actor pActor){
	try{
		if(esta(pActor)){
			getMilista().remove(pActor);
		}else{
			System.out.println("El actor no se encuentra en la lista");
		}
	}catch(NullPointerException e){
		System.out.println("El actor que desea eliminar no existe");
	}
}

public Actor buscarActor(Actor pActor){
	try{
		if(esta(pActor)){
			return this.getMilista().get(pActor.getNombre());
		}else{
			return null;
		}
	}catch(NullPointerException e){
		System.out.println("El actor que intentas buscar no existe");
		return null;
	}
}

public Actor buscarActorNombre(String pActor){
	//Devuelve el Actor si está y sino devuelve null
	return this.getMilista().get(pActor);
}

public void resetear(){
	getListaActoresPrincipal().getMilista().clear();
}

public void imprimir(){
	Iterator<String> it = getListaActoresPrincipal().getIterador();
	while(it.hasNext()){
		getListaActoresPrincipal().getMilista().get(it.next()).imprimirInformacion();
		System.out.println("\n\n");  //Imprime dos lineas vacías.
	}
}

public void ordenarLista(){
	Actor[] milistaordenada = this.convertirHashArray();
	ordenacionPorBurbuja(milistaordenada);
	//JOptionPane.showMessageDialog(null, "El resultado se muestra por consola");
	for(int i = 0; i<milistaordenada.length; i++)
		System.out.println(i+": "+milistaordenada[i].getNombre());
}

public Actor[] convertirHashArray(){
	Actor[] miArrayDeActores = new Actor[this.getMilista().size()];
	int i = 0;
	Iterator<String> itr = this.getIterador();
	while (itr.hasNext()){
		miArrayDeActores[i] = this.getMilista().get(itr.next());
		i++;
	}
	return miArrayDeActores;
}

public void ordenacionPorBurbuja(Actor[] tabla) {
	int out, in;
	for (out = tabla.length - 1; out > 0; out--)
```

```
                        for (in = 0; in < out; in++)
                                if ( tabla[in].compareTo(tabla[in + 1]) > 0 )
                                        swap(tabla, in, in + 1);
        }

        private void swap(Actor[] tabla, int a, int b) {
                Actor aux = tabla[a];
                tabla[a] = tabla[b];
                tabla[b] = aux;
        }
}
```

5.1.7 Clase ListaPeliculasPrincipal

```
package lab3;

import java.util.HashMap;
import java.util.Iterator;

public class ListaPeliculasPrincipal {
        //Atributos
        private static ListaPeliculasPrincipal milistaprincipaldepeliculas = new ListaPeliculasPrincipal();
        private HashMap<String, Pelicula> lista;


        //Constructora
        public ListaPeliculasPrincipal(){
                this.lista = new HashMap<String, Pelicula>();
        }


        //Getters y Setters
        public static ListaPeliculasPrincipal getListaPeliculasPrincipal(){
                return milistaprincipaldepeliculas;
        }


        private HashMap<String, Pelicula> getLista(){
                return this.lista;
        }

        public Iterator<String> getIterador(){
                return this.getLista().keySet().iterator();  //Esto crea un iterador de keys o llaves

//en nuestro caso titulos de peliculas.
        }


        //Otros Metodos
        public boolean estaPelicula(Pelicula pPelicula){
                return this.getLista().containsKey(pPelicula.getTitulo());
        }

        public void anadirPelicula(Pelicula pPelicula){
                if((pPelicula!=null)&&(!estaPelicula(pPelicula))){
                        this.getLista().put(pPelicula.getTitulo(), pPelicula);
                }else{
                        System.out.println("la pelicula introducida no es valida.");
                }
        }

        public void eliminarPelicula(Pelicula pPelicula){
                if((pPelicula!=null)&&(estaPelicula(pPelicula))){
                        this.getLista().remove(pPelicula);
                }else{
                        System.out.println("La pelicula no se encuentra en la lista.");
                }
        }

        public void resetearLista(){
                this.getLista().clear();
        }

        public Pelicula buscarPeliNombre(String pTitulo){
                //Devuelve la Película si está y sino devuelve null
```

```
                return this.getLista().get(pTitulo);
        }
}
```

5.1.8 Clase Cola

```
package lab3;

public class Cola<T> {

        private T[] cola;
        private int max;
        private int numeroelementos;
        private int primero;
        private int ultimo;

        public Cola(int pMax){
                cola = (T[]) new Object[pMax];
                max = pMax;
                primero = 0;
                numeroelementos=0;
                ultimo = -1;
        }

        public int getTamano(){
                return this.numeroelementos;
        }

        public void anadir(T pAnade){
                if(numeroelementos<max){
                        numeroelementos++;
                        if(ultimo==max-1){
                                System.out.println("true");
                                ultimo=-1;
                        }
                        ultimo++;
                        cola[ultimo]=pAnade;
                }
        }

        public T sacarPrimerElemento(){
                T elem = null;
                if(numeroelementos!=0){
                        elem=cola[primero];
                        primero++;
                        if(primero==max){
                                primero=0;
                        }
                        numeroelementos--;
                }
                return elem;
        }

        public void imprimirCola(){
                int cont = 0;
                int aux = primero;
                while(cont<numeroelementos){
                        System.out.println(cola[aux]);
                        aux++;
                        cont++;
                        if(aux==max){
                                aux=0;
                        }
                }
        }

}
```

5.1.9 Clase Pila

```
package lab3;

public class Pila<T> {

	private T[] pila;
	private int max;
	private int numeroelementos;
	private int primero;
	private int ultimo;

	public Pila(int pMax){
		pila = (T[]) new Object[pMax];
		max = pMax;
		primero = 0;
		numeroelementos=0;
		ultimo = -1;
	}

	public int getTamano(){
		return this.numeroelementos;
	}

	public void anadir(T pAnade){
		if(numeroelementos<max){
			numeroelementos++;
			if(ultimo==max-1){
				System.out.println("true");
				ultimo=-1;
			}
			ultimo++;
			pila[ultimo]=pAnade;
		}
	}

	public T sacarPrimerElemento(){
		T elem = null;
		if(numeroelementos!=0){
			elem=pila[ultimo];
			ultimo--;
			if(ultimo==max){
				ultimo=0;
			}
			numeroelementos--;
		}
		return elem;
	}
}
```

5.1.10 Clase StopWatch

```
package lab3;

public class StopWatch {

	private final long start;

	public StopWatch(){
		start=System.currentTimeMillis();
	}
	public double elapsedTime(){
		long now = System.currentTimeMillis();
		return (now-start)/1000.0;
	}
}
```

5.2 Pruebas

5.2.1 Clase PruebasCola

```
package lab3;

public class PruebasCola {

	public static void main(String[] args){
		Cola<Integer> c = new Cola<Integer>(3);
		System.out.println("Añado el 3");
		c.anadir(3);
		System.out.println("El tamaño de la cola tiene que ser 1 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Añado el 6");
		c.anadir(6);
		System.out.println("El tamaño de la cola tiene que ser 2 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Añado el 23");
		c.anadir(23);
		System.out.println("El tamaño de la cola tiene que ser 3 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Añado el 567");
		c.anadir(567);
		System.out.println("El tamaño de la cola tiene que ser 3 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Saco el primero");
		c.sacarPrimerElemento();
		System.out.println("El tamaño de la cola tiene que ser 2 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Añado el 400");
		c.anadir(400);
		System.out.println("El tamaño de la cola tiene que ser 3 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Saco el primero");
		c.sacarPrimerElemento();
		System.out.println("El tamaño de la cola tiene que ser 2 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Saco el primero");
		c.sacarPrimerElemento();
		System.out.println("El tamaño de la cola tiene que ser 1 y es: "+c.getTamano());
		c.imprimirCola();

		System.out.println("Añado el 4142");
		c.anadir(4142);
		System.out.println("El tamaño de la cola tiene que ser 2 y es: "+c.getTamano());
		c.imprimirCola();
	}

}
```

5.2.2 Clase Pra

```
package lab3;

import java.io.File;
```

```
public class Pra {

	public static void main(String[] args){
		File archivo = new File("C:\\Documents and Settings\\HP_Propietario\\Es-
critorio\\aaaa.txt");
		Main.getMain().cargarFichero(archivo);
		String[] aaa = Main.getMain().rellenarArray();
		for(String s:aaa){
			System.out.println(s);
		}
	}
}
```

5.2.3 Clase PruebaEstanRelacionados

```
package lab3;

import java.io.File;

public class PruebaEstanRelacionados {

	public static void main(String[] args){

		//Utilizo el fichero que esta en el src
		File archivo = new File("C:\\Users\\Mikel\\Desktop\\actresses-small-ok.txt");
		Main.getMain().cargarFichero(archivo);

		Actor a1 = null;
		Actor a2 = null;
		Pila<Actor> p = new Pila<Actor>(20);


		System.out.println("Prueba estanRelacionados (distancia)");

		System.out.println("Prueba 1:");
		System.out.println("Vamos a hallar la distancia entre Aaliste, Agnes y Aalders, Mairen que
tiene que ser 0");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aaliste, Agnes");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalders, Mairen");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionados(a1, a2));
		}else{
			System.out.println("FATAL ERROR 1");
		}


		System.out.println("Prueba 2:");
		System.out.println("Vamos a hallar la distancia entre Aalam Roxanne y Aalbers Francisca que
tiene que ser 1");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Roxanne");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalbers, Fran-
cisca");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionados(a1, a2));
		}else{
			System.out.println("FATAL ERROR 2");
		}


		System.out.println("Prueba 3:");
		System.out.println("Vamos a hallar la distancia entre Aalam Roxanne y Aalan, Lynne que tiene
que ser 2");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Roxanne");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalan, Lynne");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionados(a1, a2));
		}else{
			System.out.println("FATAL ERROR 3");
		}
```

```
		System.out.println("Prueba 4:");
		System.out.println("Vamos a hallar la distancia entre Aalbregt, Djamilla y Aalia, Nila que tiene que ser 3");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalbregt, Djamilla");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalia, Nila");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionados(a1, a2));
		}else{
			System.out.println("FATAL ERROR 4");
		}



		System.out.println("Prueba 5:");
		System.out.println("Vamos a hallar la distancia entre Aaliste, Agnes y Aalam, Schahla que tiene que ser 4");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aaliste, Agnes");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Schahla");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionados(a1, a2));
		}else{
			System.out.println("FATAL ERROR 5");
		}

		System.out.println();
		System.out.println();
		System.out.println("Prueba estanRelacionados(boolean)");

		System.out.println("Prueba 6: Vamos a comprobar si Aalam, Schahla y Aalam, Roxanne son colegas, ha de responder true");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Schahla");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Roxanne");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionadosBool(a1, a2));
		}else{
			System.out.println("FATAL ERROR 6");
		}


		System.out.println("Prueba 7: Vamos a comprobar si Aalam, Schahla y Aalgaard, Leannie son colegas, ha de responder false");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Schahla");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalgaard, Leannie");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionadosBool(a1, a2));
		}else{
			System.out.println("FATAL ERROR 7");
		}


		System.out.println("Prueba 8: Vamos a comprobar si Aalia, Nila y Aalbu, Fern son colegas, ha de responder true");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalia, Nila");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalbu, Fern");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionadosBool(a1, a2));
		}else{
			System.out.println("FATAL ERROR 8");
		}


		System.out.println("Prueba 9: Vamos a comprobar si Aalbu, Courtney y Aalam, Roxanne son colegas, ha de responder false");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalbu, Courtney");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Roxanne");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionadosBool(a1, a2));
		}else{
			System.out.println("FATAL ERROR 9");
		}


		System.out.println("Prueba 10: Vamos a comprobar si Aalbers, Francisca y Aaliste, Agnes son colegas, ha de responder true");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalbers, Francisca");
```

```
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aaliste, Agnes");
		if((a1!=null)&&(a2!=null)){
			System.out.println(Main.estanRelacionadosBool(a1, a2));
		}else{
			System.out.println("FATAL ERROR 10");
		}

		System.out.println();
		System.out.println();
		System.out.println("Prueba estanRelacionados (distancia y nombres)");


		System.out.println("Prueba 11: Vamos a hallar el camino entre Aalam Roxanne y Aalbers Francisca y el resultado ha de ser vacío porque no hay actrices de por medio");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalbers, Francisca");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Roxanne");
		if((a1!=null)&&(a2!=null)){
			p = Main.estanRelacionadosNombres(a1, a2);
			while(p.getTamano()>0){
				System.out.print("-"+p.sacarPrimerElemento().getNombre());
			}
			System.out.println();
		}else{
			System.out.println("FATAL ERROR 11");
		}

		System.out.println("Prueba 12: Vamos a hallar el camino entre Aalam Roxanne y Aalan, Lynne y el resultado ha de ser: Aalam, Schahla");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalan, Lynne");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Roxanne");
		if((a1!=null)&&(a2!=null)){
			p = Main.estanRelacionadosNombres(a1, a2);
			while(p.getTamano()>0){
				System.out.print("-"+p.sacarPrimerElemento().getNombre());
			}
			System.out.println();
		}else{
			System.out.println("FATAL ERROR 12");
		}

		System.out.println("Prueba 13: Vamos a hallar el camino entre Aalbregt, Djamilla y Aalia, Nila y el resultado ha de ser: Aalam, Roxanne-Aalbers, Francisca");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalbregt, Djamilla");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalia, Nila");
		if((a1!=null)&&(a2!=null)){
			p = Main.estanRelacionadosNombres(a1, a2);
			while(p.getTamano()>0){
				System.out.print("-"+p.sacarPrimerElemento().getNombre());
			}
			System.out.println();
		}else{
			System.out.println("FATAL ERROR 13");
		}


		System.out.println("Prueba 14: Vamos a hallar el camino entre Aaliste, Agnes y Aalam, Schahla y el resultado ha de ser: Aalia, Nila-Aalbers, Francisca-Aalam, Roxanne");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aaliste, Agnes");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalam, Schahla");
		if((a1!=null)&&(a2!=null)){
			p = Main.estanRelacionadosNombres(a1, a2);
			while(p.getTamano()>0){
				System.out.print("-"+p.sacarPrimerElemento().getNombre());
			}
			System.out.println();
		}else{
			System.out.println("FATAL ERROR 13");
		}


		System.out.println("Prueba 15: Vamos a hallar el camino entre Aaliste, Agnes y Aalan, Lynne y el resultado ha de ser: Aalia, Nila-Aalbers, Francisca-Aalam, Roxanne-Aalam, Schahla");
		a1 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aaliste, Agnes");
		a2 = ListaActoresPrincipal.getListaActoresPrincipal().buscarActorNombre("Aalan, Lynne");
		if((a1!=null)&&(a2!=null)){
```

```
                        p = Main.estanRelacionadosNombres(a1, a2);
                        while(p.getTamano()>0){
                                System.out.print("-"+p.sacarPrimerElemento().getNombre());
                        }
                        System.out.println();
                }else{
                        System.out.println("FATAL ERROR 13");
                }
        }
}
```

5.2.4 Clase PruebaGradoRelaciones

```
package lab3;

import java.io.File;

public class PruebaGradoRelaciones {

        public static void main(String[] args){
                File archivo = new File("C:\\Documents and Settings\\HP_Propietario\\Es-
critorio\\aaaa.txt");
                Main.getMain().cargarFichero(archivo);
                System.out.println(Main.getMain().gradoRelaciones());
        }

}
```

5.2.5 Clase Pruebas

```
package lab3;

import java.util.ArrayList;

public class Pruebas {
        public void pruebaPos(){
                ArrayList<Actor> miLista = new ArrayList<Actor>();
                Actor a1=new Actor("Pedro");
                Main.getMain().buscarPosMin(miLista);
        }
}
```

6 Conclusiones

En estos laboratorios hemos invertido bastante tiempo y hemos estado trabajando un buen número de horas para hacerlo funcionar. Hemos aprendido a trabajar con ficheros de texto, implementar interfaces, y también hemos hecho uso de nuevas estructuras de datos que hasta la fecha desconocíamos, como por ejemplo pilas, colas, tablas hash, listas ligadas…

Esto nos ha servido para aplicar lo aprendido en clase y en ocasiones, para buscar información por nuestra cuenta.

Los laboratorios no han tenido como único objetivo la implementación de un programa que cumpla lo especificado, si no también que lo haga de una forma eficiente. Es por eso que en cada método hemos usado la estructura de datos que más se adecúa a la función.